**Asymptotic notation**

Asymptotic Notation is **used to describe the running time of an algorithm** - how much time an algorithm takes with a given input, n. There are three main different notations: **big O, big Theta (Θ), and big Omega (Ω).**

Let’s say

**n = number of inputs given to a function**,

**g(n) = a standard time (like log n, 1, n, n! etc. to which the actual time can be broken down)**,

**C = constant time taken by the function**

***Big oh (O)***

The Big O notation defines an **upper bound of an algorithm**, it bounds a function only from above, it describes the **worst-case running time** of a program.

So, **f(n) = O(g(n)) says f(n) <= C.g(n)**

**Big Oh – measuring run time by counting the number of steps an algorithm takes (translating to actual time is trivial).**

BigO

Example, A linear search takes O(n) times (if the element is at the last position which is worst case), so

f(n) = O(g(n)), here g(n) = n, means c \* n is upper bound of f(n)

**f(n) = O(g(n)) means c\*g(n) is an upper bound on f(n)**

***Big Omega(Ω)***

It describes the **best running time** of a program. It defines **lower bound** of an algorithm.

So, f(n) = Ω (g(n)) says f(n) >= C.g(n)

Example, A linear search takes Ω (1) (if the element is at first which is best case)

BigOmega

**f(n) = Ω(g(n)) means c\*g(n) is a lower bound on f(n)**

***Big-Θ***

It describes the **exact time case / average time case** of a function. It defines **tight bound of a function.**

So, f(n) = Θ (g(n)) says c1\*g(n) <= f(n) <= c2\*g(n)

thetanotation

**f(n) = Θ(g(n)) means c1\*g(n) is an upper bound on f(n) and c2\*g(n) is a lower bound on f(n)**

**Algorithmic Common Runtimes**

The common algorithmic runtimes from **fastest to slowest** are:

* constant: Θ(1)
* logarithmic: Θ(log N)
* linear: Θ(N)
* polynomial: Θ(N^2)
* exponential: Θ(2^N)
* factorial: Θ(N!)

**1 < log(n) < n < n\*log(n) < n^2 < n^3 < 2^n < n!**